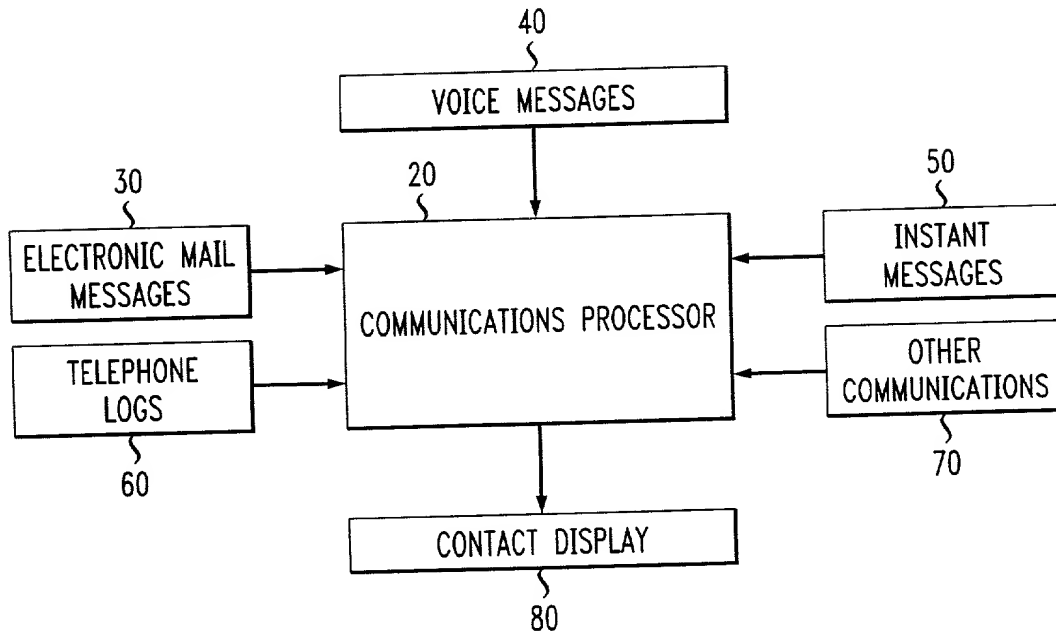
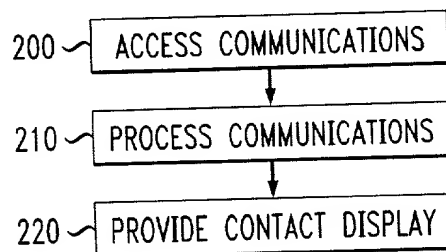
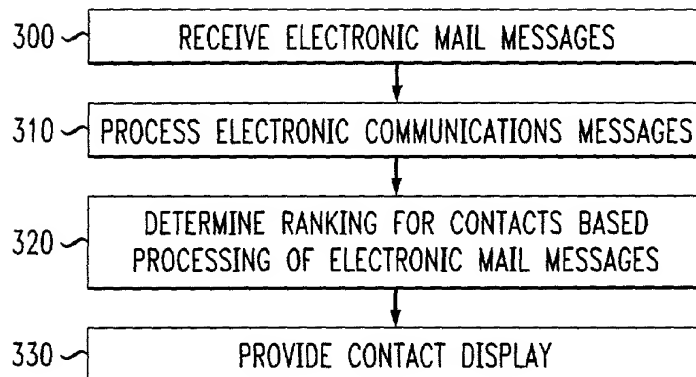
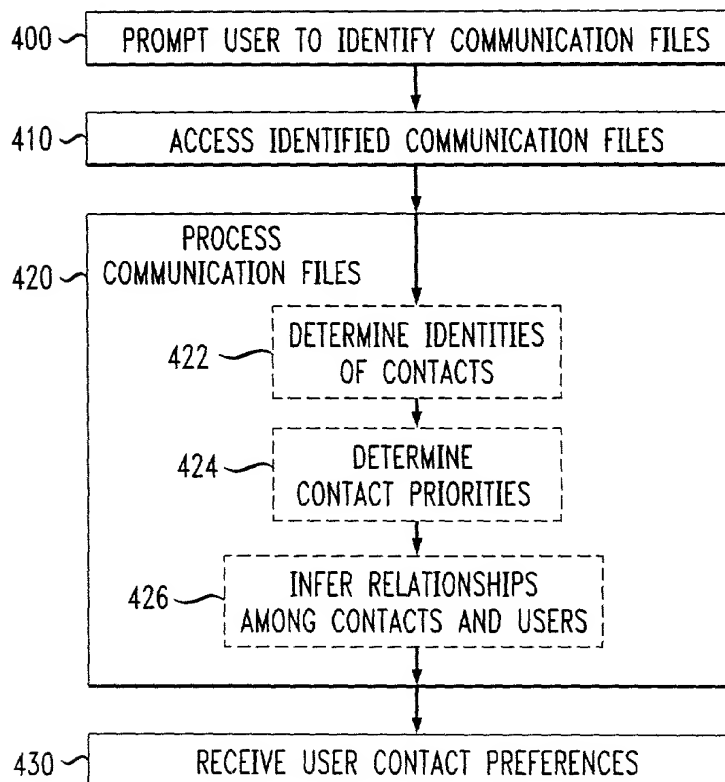


1/16

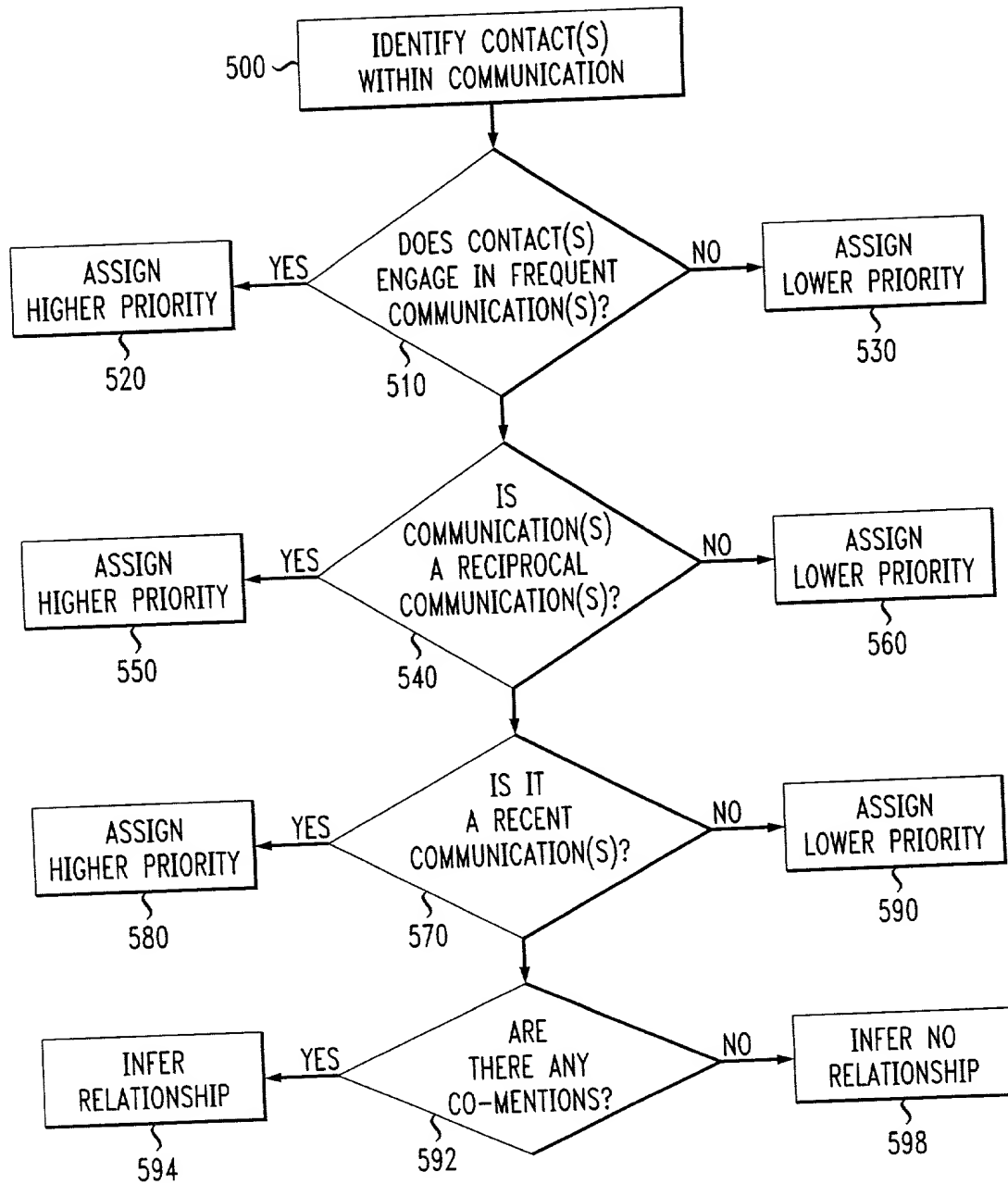
FIG. 1*FIG. 2*

2/16

FIG. 3*FIG. 4*

3/16

FIG. 5



4/16

FIG. 6

600

ContactMap: Email Analysis Results						
Email Analysis Contact View Help						
<input type="button" value="Add To Contact Network"/>		<input type="button" value="Mark All"/>				
Include?	Name	Address	Oldest	Newest	Score	
<input type="checkbox"/>	Jacob Bardam	bardram@dk.ibm.com	02-Dec-...	02-Dec-...	101	▲
<input type="checkbox"/>	T.L. Taylor	ttaylor@binah.cc.brandeis.edu	02-Dec-...	02-Dec-...	101	
<input type="checkbox"/>	Cornwell, Patricia	cornwell@exch.hpl.hp.com	02-Dec-...	02-Dec-...	202	
<input type="checkbox"/>	Seacliff Secritarial	ccliff@flash.net	03-Dec-...	03-Dec-...	101	
<input type="checkbox"/>	Raven Earlygrow	anima@mcn.org	04-Dec-...	04-Dec-...	101	
<input type="checkbox"/>	Weller, Susan C.	sweller@utmb.edu	08-Dec-...	08-Dec-...	101	
<input type="checkbox"/>	Dr. Yair Neuman	yneuman@bgumail.bgu.ac.il	19-Jan-99	19-Jan-99	101	
<input type="checkbox"/>		Michael_Muller/CAM/@lotus.com	19-Jan-99	19-Jan-99	101	
<input type="checkbox"/>	Mark Goldstein	kvetch@worldnet.att.net	21-Jan-99	21-Jan-99	101	
<input type="checkbox"/>	Jean Scholtz	jean.scholtz@nist.gov	27-Feb-...	27-Feb-...	101	
<input type="checkbox"/>		scholtz@zing.ncsl.nist.gov	27-Feb-...	27-Feb-...	2	
<input type="checkbox"/>	Gitte Wind Johans.	gitte_johansen@image.dk	28-Feb-...	28-Feb-...	101	
<input type="checkbox"/>	Dr.Dr.Norbert Streitz	streitz@darmstadt.gmd.de	25-Mar-99	25-Mar-99	101	
<input type="checkbox"/>		Andrew_L_Cohen/CAM/@lotus.com	02-Mar-99	02-Mar-99	101	
<input type="checkbox"/>		jjohnson@uiwizards.com(jeff johnson)	04-Mar-99	04-Mar-99	101	
<input type="checkbox"/>	Yin Yin Wong	yinyin@dnai.com	05-Mar-99	05-Mar-99	101	
<input type="checkbox"/>		Sbarley@leland.stanford.edu	05-May-...	05-May-...	3	
<input type="checkbox"/>	Katie Hafner	hafner@nytimes.com	06-May-...	06-May-...	101	
<input type="checkbox"/>	Katherine Innis	kinnis@mit.edu	06-May-...	06-May-...	101	
<input type="checkbox"/>	Aileen Broccardo	aileen@ics.uci.edu	14-Jun-99	14-Jun-99	101	
<input type="checkbox"/>	Susi Skomal	SSKOMAL@aaanet.org	14-Jun-99	14-Jun-99	101	
<input type="checkbox"/>	Frank Cervarich	ccc@cais.com	14-Jun-99	14-Jun-99	101	
<input type="checkbox"/>		dsjohnson@att.com	14-Jun-99	14-Jun-99	101	
<input type="checkbox"/>	E-care	E-care@greenmountian.com	15-Jun-99	15-Jun-99	101	
<input type="checkbox"/>	Syed Shariq	sshariq@stanford.edu	21-Jul-99	21-Jul-99	1	▼
660	610	620	630	640	650	

09726912, 020806

5/16

FIG. 7A

```

/* --Java--
*****
*
* File:      MATContactCounts.java
* RCS:      $Header: $
* Description:
* Author:    John Hainsworth
* Created:   Fri Sep 03 15:17:42 1999
* Modified:  Thu Sep 09 16:15:41 1999 (Michael L. Creech) mike@home
* Language:  Java
* Package:   com.att.research.mat.email
* Status:    Experimental (Do Not Distribute)
*
* (c) Copyright 1999, AT&T, all rights reserved.
*
*****
*
* Revisions:
*
* Thu Sep 09 16:15:31 1999 (Michael L. Creech) mike@home
*   Added getOldest( ), and getNewest( ).
* Fri Sep 03 15:17:46 1999 (Michael L. Creech) mike@home
*   Made getOrderingValue( ) public.
*****
*/

package com.att.research.mat.email;

import java.io.PrintWriter;
import java.io.IOException;
import java.util.Date;

/** Statistics about mail to and from one e-mail address.
 * These statistics are all redundant with respect to the data,
 * so none of them are saved with the data.
 */

```

09726942-020802

6/16

FIG. 7A continued

```

public class MATContactCounts {
    /** Copyright (c) 1999 AT&T labs */
    static public String Copyright = "Copyright (c) 1999 AT&T Labs";

    static final private int CountNONE = -1;
    static final private double DCountNONE = -1.0;

    static final int FromME = 0;
    static final int FromOTHER = FromME + 1;
    static private final int From_N = FromOTHER + 1;
    static private final String StrFrom[ ] = { "My", " " };

    /** seen in an original message */
    static final int KindCOMPOSITION = 0;
    /** seen in a reply message */
    static final int KindREPLY = KindCOMPOSITION + From_N;
    /** seen in a forwarded message */
    static final int KindFORWARD = KindREPLY + From_N;
    static private final int Kind_N = KindFORWARD + From_N;
    static private final String StrKind[ ] = { " ", "Repl", "Fwd" };

    /** seen in a "To:" header */
    static final int WhereTo = 0;
    /** seen in a "From:" header */
    static final int WhereFROM = WhereTo + Kind_N;
    /** seen in a "Cc:" header */
    static final int WhereCC = WhereFROM + Kind_N;
    /** seen in a "Bcc" header */
    static final int WhereBcc = WhereCC + Kind_N;
    static private final int Where_N = WhereBCC + Kind_N;
    static private final String StrWhere[ ] = { "To", "From", "Cc", "Bcc" };

```

00726944 020502

7/16

FIG. 7B

```

// Java will initialize these to 0
private int counts[ ] = new int [Where_N];

Date newest;
Date oldest;

public Date getOldest ( )
{
    return oldest;
}

public Date getNewest( )
{
    return newest;
}

/** @param what Should always be of the form (From?? + Kind?? + Where??) */
void incrementCount(Date d, int what) {
    counts[what]++;
    if (null != d) {
        if (null == newest) {
            oldest = newest = d;
        } else {
            long nms = d.getTime( );
            if (nms > newest.getTime( ) ) {
                newest = d;
            } else if (nms < oldest.getTime( ) ) {
                oldest = d;
            }
        }
    }
}

/** @param what Should always be of the form (From?? + Kind?? + Where??) */
int getCount(int what) {
    return counts[what];
}

```

8/16

FIG. 7B continued

```

/** Set all counters to zero. */
void clear( ) {
    int i = 0;
    while (i < Where_N) {
        counts[i++] = 0;
    }
    nonZeroTotal = CountNONE;
    orderingValue = CountNONE;
    oldest = newest = null;
    duration = CountNONE;
    density = CountNONE;
    freshness = DCountNONE;
}

/** Get total number of messages this contact actually appeared in.
 * Since we add To:s to Cc:s and Cc:s to Bcc:s, we just need to count
 * From:s and Bcc:s. */
private int getMessageCount( ) {
    return (counts [FromME + KindCOMPOSITION + WhereFROM] +
            counts [FromME + KindREPLY + WhereFROM] +
            counts [FromME + KindFORWARD + WhereFROM] +
            counts [FromOTHER + KindCOMPOSITION + WhereFROM] +
            counts [FromOTHER + KindREPLY + WhereFROM] +
            counts [FromOTHER + KindFORWARD + WhereFROM] +
            counts [FromME + KindCOMPOSITION + WhereBCC] +
            counts [FromME + KindREPLY + WhereBCC] +
            counts [FromME + KindFORWARD + WhereBCC] +
            counts [FromOTHER + KindCOMPOSITION + WhereBCC] +
            counts [FromOTHER + KindREPLY + WhereBCC] +
            counts [FromOTHER + KindFORWARD + WhereBCC]);
}

```

00726910 00000000

9/16

FIG. 7C

```

private long duration = CountNONE;
public long getDuration( ) {
    if (CountNONE == duration) {
        if (null != oldest) {
            duration = newest.getTime( ) - oldest.getTime( );
        } else {
            duration = 0;
        }
    }
    return duration;
}

private long density = CountNONE;
public long getDensity( ) {
    if (CountNONE == density) {
        long dur = getDuration( );
        if (0 < dur) {
            density = getMessageCount( ) / dur;
        } else {
            density = 0;
        }
    }
    return density;
}

private double freshness = DCountNONE;
public double getFreshness( ) {
    if (DCountNONE == freshness) {
        freshness = { ( (double) getDensity( ) ) /
                      ( (double) ( (new Date( ) ).getTime( ) - newest.getTime( ) ) ) );
    }
    return freshness;
}

```

09756912 020606

10/16

FIG. 7C continued

```

private int noneZeroTotal = CountNONE;
int getNonZeroTotal ( ) {
    if (CountNONE == noneZeroTotal) {
        noneZeroTotal = 0;
        int i = 0;
        while (i < Where_N) {
            noneZeroTotal += counts [i];
            i += 1;
        }
    }
    return noneZeroTotal;
}

private int orderingValue = CountNONE;

// MLC:
public int getOrderingValue ( ) {
    if (CountNONE == orderingValue) {
        ordeingValue =
            counts [FromME + KindREPLY + WhereTO] * 1000000 +
            counts [FromME + KindCOMPOSITION + WhereTO] * 10000 +
            counts [FromOTHER + KindREPLY + WhereFROM] * 100 +
            counts [FromOTHER + KindCOMPOSITION + WhereFROM] * 100 +
            getNonZeroTotal ( );
    }
    return orderingValue;
}

```

11/16

FIG. 7D

```

private String print1 (String leader, PrintWriter ps, int what)
    throws IOException {
    if (0 != counts [what]) {
        int iFrom = what % From_N;
        int iKind = (what % Kind_N) / From_N;
        int iWhere = what / Kind_N;
        ps.print (leader+StrFrom[iFrom]+StrKind[iKind]+StrWhere[iWhere]+
            "=" + counts[what] );
        leader = " ";
    }
    return leader;
}

public void print (PrintWriter ps, int format) throws IOException {
    switch (format) {
    case MATData.FormatANALYSIS:
        String leader = " {";
        int what = 0;
        leader = print1 (leader, ps, FromME + KindREPLY + WhereTO);
        leader = print1 (leader, ps, FromME + KindCOMPOSITION + WhereTO);
        leader = print1 (leader, ps, FromOTHER + KindREPLY + WhereFROM);
        leader = print1 (leader, ps, FromOTHER + KindCOMPOSITION + WhereFROM);
        while (what < Where_N) {
            switch (what) {
            case FromME + KindREPLY + WhereTO:
            case FromME + KindCOMPOSITION + WhereTO:
            case FromOTHER + KindREPLY + WhereFROM:
            case FromOTHER + KindCOMPOSITION + WhereFROM:
                break;
            default:
                leader = print1 (leader, ps, what);
                break;
            }
            what++;
        }
        if (leader.equals(" ") ) {
            ps.print ("{}");
        }
        break;
    }
}

```

09766916 000000

12/16

FIG. 7D continued

```
case MATData.FormatTERMINAL:
    int what2 = 0;
    String leader2 = " ";
    while (what2 < Where_N) {
        leader = print1 (leader2, ps, what2++);
    }
    if (leader2.equals (" ") ) {
        ps.print ("{}");
    }
    ps.print (" score="+getOrderingValue ( ) );
    break;
case MATData.FormatDATABASE:
    break;
}
}
```

13/16

FIG. 8A

```

package com.att.research.mat.email;

import com.att.research.mat.utility.UniqueIDs;
import com.att.research.mat.utility.NumberedMember;
import java.util.Enumeration;
import java.io.PrintWriter;
import java.io.IOException;
import java.io.StringWriter;
import java.util.Date;

/** A list of contacts that appeared together on a To, Cc, or Bcc line. */

public class MATContactGroup extends UniqueIDs {
    /** Copyright (c) 1999 AT&T Labs */
    static public String Copyright = "Copyright (c) 1999 AT&T Labs";

    /** @serial cc not stored */
    private MATContacts cc;

    // MLC:
    public MATContacts getMATContacts ( )
    {
        return cc;
    }

    public MATContactGroup (MATContacts cc) {
        this.cc = cc;
    }

    public Enumeration elements ( ) { return elements (cc); }

    public void hideUnimportantMembers (int threshold) {
        Enumeration e = elements (cc);
        while (e.hasMoreElements ( ) ) {
            ((MATContact) e.nextElement ( ) ).hideIfUnimportant (threshold);
        }
    }
}

```

FIG. 8A continued

```

public boolean containsMe ( ) {
    Enumeration e = elements (cc);
    while (e.hasMoreElements ( ) ) {
        if ( ( (MATContact) e.nextElement ( ) ) .isMe ( ) ) {
            return true;
        }
    }
    return false;
}

/** @return Unique id of removed member. */
public int removeMe ( ) {
    int i = 0;
    Enumeration e = elements (cc);
    while (e.hasMoreElements ( ) ) {
        MATContact c = (MATContact) e.nextElement ( );
        if (c.isMe ( ) ) {
            removeElementAt (i);
            return c.getUniqueId ( );
        }
        i++;
    }
    return NumberedMember.IdNONE;
}

```

```

public MATContactGroup updateToRemoveMe ( ) {
    int removedMe = removeMe ( );
    if (NumberedMember.IdNONE != removedMe) {
        int others = toMATContactID ( );
        MATContactGroup meGroup = new MATContactGroup(cc);
        meGroup.addElement (removedMe);
        meGroup.addElement (others);
        meGroup.sortIDs ( );
        return meGroup;
    } else {
        return null;
    }
}

public int toMATContactID ( ) {
    switch (size( ) ) {
        case 0:
            return NumberedMember.IdNONE;
        case 1:
            return uniqueIdAT (0);
        default:
            int removedMe = removeMe ( );
            if (NumberedMember.IdNONE != removedMe) {
                int others = toMATContactID ( );
                MATContactGroup meGroup = new MATContactGroup(cc);
                meGroup.addElement(removedMe);
                meGroup.addElement(others);
                meGroup.sortIDs ( );
                MATContact c = cc.find (meGroup);
                c.setHidden(true);
                return c.getUniqueID ( );
            } else {
                sortIds ( );
                return cc.find(this).getUniqueID ( );
            }
    }
}
}

```

FIG. 8B continued

```

/** Calls {@link com.att.research.mat.email.MATContact#incrementCount(Date, int)}
 * for each member. */
void incrementCount(Date d, int what) {
    Enumeration e = elements(cc);
    while (e.hasMoreElements( )) {
        ((MATContact)e.nextElement( )).incrementCount(d, what);
    }
}

/** Needed by the contact the viewer. */
// This method MUST use cc, and cannot get it as an argument.
// This is the only true reason that cc must be a member of this object.
public String toString( ) {
    StringWriter sw = new StringWriter( );
    PrintWriter ps = new PrintWriter(sw);
    try {
        print(ps, cc, " ", MATData.FormatADDRESS_ONLY);
    } catch (IOException e) {
    }
    return sw.toString( );
}
}

```